CSC309 笔记

2016年5月16日 12:48

Week2:

Review:

- SYN-SYN-ACK?
  - How computers(client server) communicate

    3-way handshake
    - 要先走一遍流程才能连接

    What's packet => formatted unit of data
- Github
  - https://rogerdudler.github.io/git-guide/

HTML5 & CSS3 (talk in general sense)

- HTML5
  - Support an older browser => modernizr => support the new HTML5 and CSS3 elements and ignore it.

    Semantic HTML => a semantic approach to name tags/ elements that represent the info you provides
    Eg: <em> is about content(emphasize) rather than <i> about style(italic)
    Why care? Too much divs! Semantic makes more sense, better readability, interoperability (div can be named randomly, semantic element is stable)

    Audio and video control

    <canvas> => draw graphics, use script to draw

    Drag and drop

    Web storage => store on client computer, cookies, simple to use

    IndexedDB (indexed database) => complicate asynchronous(processed outside of the main program flow) API, deal better with complex data

    Server-sent events => not every browser support, just use JavaScript(jQuery)

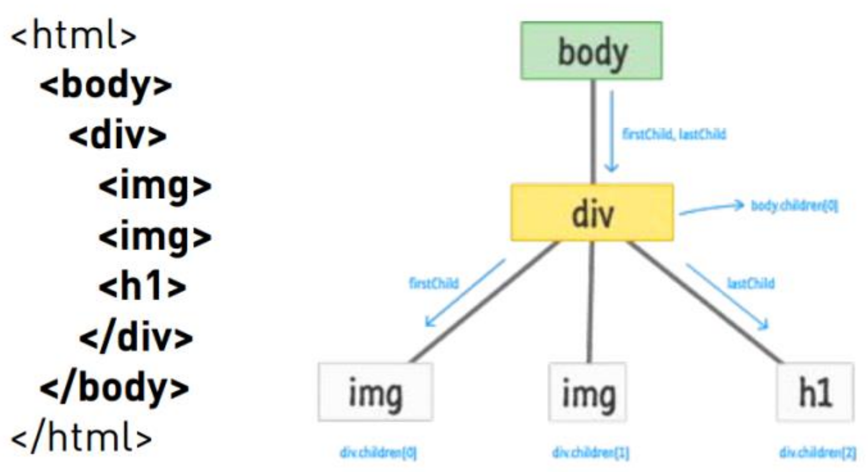
    Geological API

- CSS3
  - Most modern browsers support standards version

    Viewport => visible area of the website seen through the browser window
    - Style depends on different size

DOCUMENT OBJECT MODEL (the dom)

- Web browser builds a hierarchical model of the web page that includes all of the objects in the page and their relationship to each other.
  All of properties, methods and events used to manipulate and create web pages are organized into objects

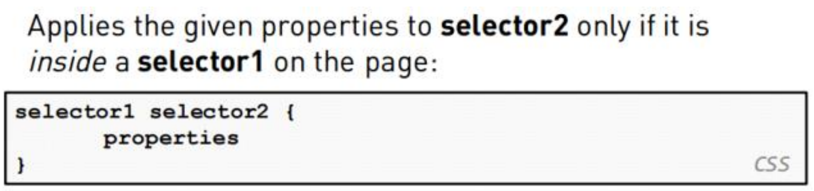


```
<html>
  <body>
    <div>
      <img>
      <img>
      <h1>
    </div>
  </body>
</html>
```

Use scripting languages to access objects in the DOM model
Change object in response to events, including user requests

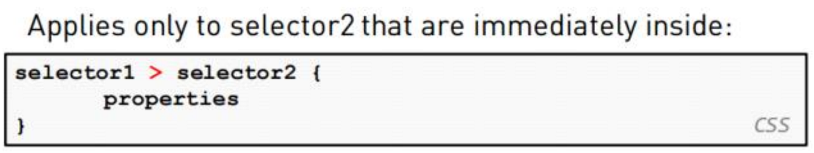
PAGE LAYOUT

<section><article><aside><div> => logical section area, no appearance
<span> => inline section



Applies the given properties to **selector2** only if it is *inside* a **selector1** on the page:

```css
selector1 selector2 {
        properties
}
```
CSS

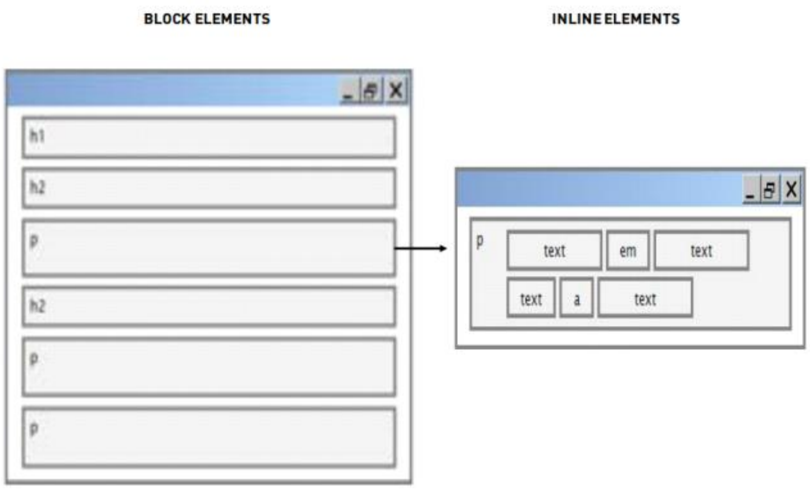


Applies only to selector2 that are immediately inside:

```css
selector1 > selector2 {
        properties
}
```
CSS

如果装了插件导致selector冲突，则可以重新命名selector，比如加前缀来避免冲突,but its not semantic

Content/padding/border/margin
    Default styles applied => e.g.: left margin
    Block => inline



Border: visually separate content
Padding: let element content breath, inside
Margin: exists around the element, outside

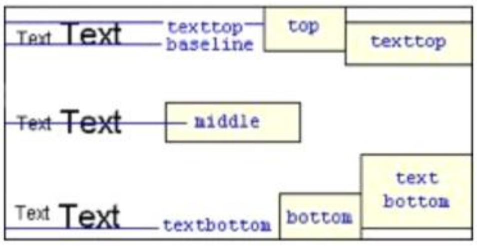
Layout order:

1. try aligning the element's content

    Inline

    Text-align? => 文字横向位置，据左中右

    Vertical-align => 文字位置在上标还是下标



    Block ? What does block exactly mean?

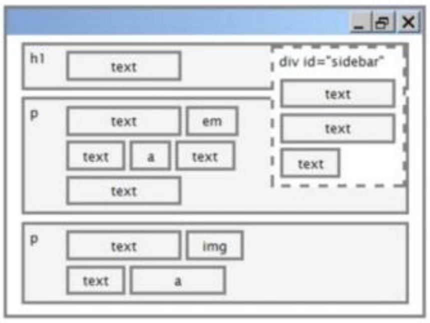
    Inline-block

    Display vs visibility

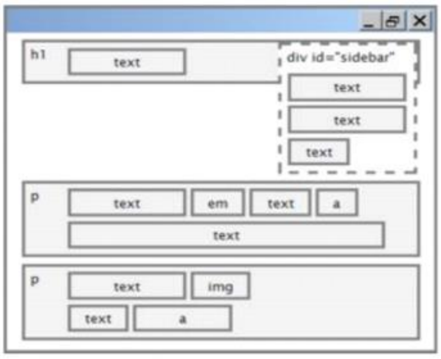
    Flexbox => block level element to the center

2. try floating the element

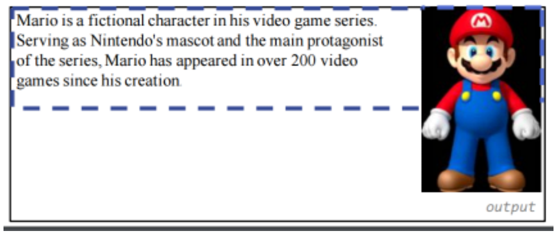
    Float: removes an element from the normal document flow, will overlap



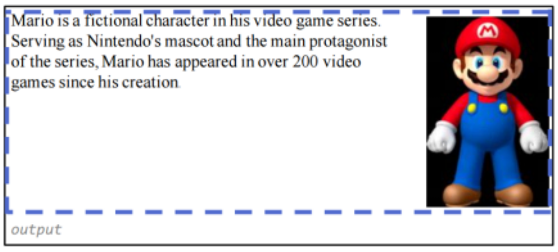
    Clear: clear elements follow a floating element to prevent wrapping/overlapping



    Overflow => deal with oversize content, 把原本overflow的东西包进去



Mario is a fictional character in his video game series. Serving as Nintendo's mascot and the main protagonist of the series, Mario has appeared in over 200 video games since his creation.



Mario is a fictional character in his video game series. Serving as Nintendo's mascot and the main protagonist of the series, Mario has appeared in over 200 video games since his creation.

Auto-cleared container?

3. try positioning the element

Absolute:

Relative:

Fixed: 无论如何调整浏览器，fixed的东西总在同样的地方显示

Difference3 between these three ones

Responsive design

Optimal experience regardless of device and screen resolution

Framework vs create your own

Media queries

Mobile first vs desktop first

Decide a breakpoint

Week 4

JavaScript

Client-side scripting (JavaScript is)

Usability => modify page without waiting for server

Efficiency => small quick changes to page

Event-driven => respond to user actions like clicks and key presses

Server-side programming (like php)

Security: can access server private data which client cannot see

Compatibility: not subject to browser compatibility issues

Power: can write files, open connections to servers, connect to databases.

JavaScript:

Make webpage interactive

Insert content dynamically into HTML/DOM

React to events

Calculation on client side

Web standard

Interpreted, not compiled

Connect to html with script tag => in head tag => but html will stop loading html and dl JS => bad experience => add async or defer(async and in order) in script tag.

JS don’t have main method, wait for user actions called events and respond to them

Eg: button

Choose control(button) and events(mouse click) => write JS function to run when the event occurs => attach function to the event on the control

JS can manipulate (change) elements on an HTML page

In JS:

Variable => var. Can be in a lot of types, JS is loosely typed language

Special value: undefined (not been declared, does not exist)/ null (exists, but null)

Logical operators:
- == is loose, automatically convert types ("5.0" == 5)
- === and !== are strict equality test check both type and value

Boolean, any value can be
- False => 0, 0.0, NaN, "", null, undefined
- Truthy => anything else
- Useful to check dynamic element exist or not

Array
- Works like C
- A lot of operations like push/ unshift(push at beginning)/ pop/ shift(pop at beginning)/ sort
- Use object as associative arrays

String
- Specified by "" or '', concatenate with+, has length property

- Can access letters of a String
  - charAt/ parseInt

- Split by using delimiter or regular expression

These are all objects:

- window => top level object in the DOM. All global code and variables becomes part of the window object properties(document, history, location, name). A lot of methods included.

- Documents => current webpage and element inside it. Can extend beyond everything

- Location => refers to URL of the current webpage

- Navigator => info about web browser application

- Screen => info about client's display screen; what visible to you, device screen

- History => keeps a list of sites that browser has visited in this window

JS => DOM, use JS to change the DOM and the html source file

Different types of DOM nodes (different from tag):
1. Element nodes (HTML tag), can have children and attributes
2. Text nodes
3. comment nodes
4. white space nodes => created by the spaces between nodes/elements/tags

   Some attributes include firstChild/ lastChild/ childNodes…

JS window.onload => attach event handlers to the global event called window.onload that occurs at the moment that page is done loading

```html
<!-- look Ma, no JavaScript! -->
<button id="ok">OK</button>
                                                  HTML
```

```javascript
// called when page loads; sets up event handlers
function pageLoad() {
	document.getElementById("ok").onclick = okayClick;
}
function okayClick() {
	alert("booyah");
}
window.onload = pageLoad; // global code
                                                  JS
```

In this example: window.onload => function pageLoad => function okayClick

JS allows you to declare anonymous functions that can be stored as a variable and attached as an event handler.

```javascript
window.onload = function() {
	var okButton = document.getElementById("ok");
	okButton.onclick = okayClick;
};
function okayClick() {
	alert("booyah");
}
                                                  JS
```

"this" refers to the current object

HTML5 canvas

<canvas> => draw graphics via scripting, only container for graphics => use script to actually draw the graphics

```javascript
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
                                                  JS
```

Draw a line, circle (with path)/ border/ text with (ctx.) and clear the rectangle

Animation

Draw/ clear/ update
- Clear: setTimeout(animate, 33);
- Draw: array to save the object like a rectangle
- Update: for loop to keep updating the objects xy coordinate

===Week 5===

Jquery

Jquery would return everything as an object, so be careful using if statement
$(document).ready(function(){

Functions
- Setters => change the selection
- Getters => read the selection (read the first element in a selection)

Chain setters since jQuery setters return the selected element

jQuery does not automatically placed on the page => need to append

Filters: filter/ not/ has

jQuery stores a reference

jQuery changes the style/ presentation

jQuery => move the first item in a list to the end

.on() event handler

Event delegation (to parent elements)

Node.js

Server-side language => event-driven, I/O driven model-based runtime environment and library for developing server-side web apps using JS

Node.js => break free of the client and process on the server-side
- Pro: Persistent, realtime application/ con: computation-heavy apps

AJAX(asynchronous JavaScript and XML)

XML

JSON

======Week 6=======

REST APIs

======Week 7=======

Sample Final question

Why don't use table for layout?

1. its not semantic
2. table are tabular data, not for layout
3. slower
3. not good with CSS

React.js

JS library for building user interfaces on the web.
- 1. declarative => create views for each state of your app (?) and particular components will be automatically updated by ReacJS
- 2. component-based => build "modules " of components that you can combine into a larger UI
- 3. language-independent => does not care about the web stack you are using, change ReactJs code as needed.

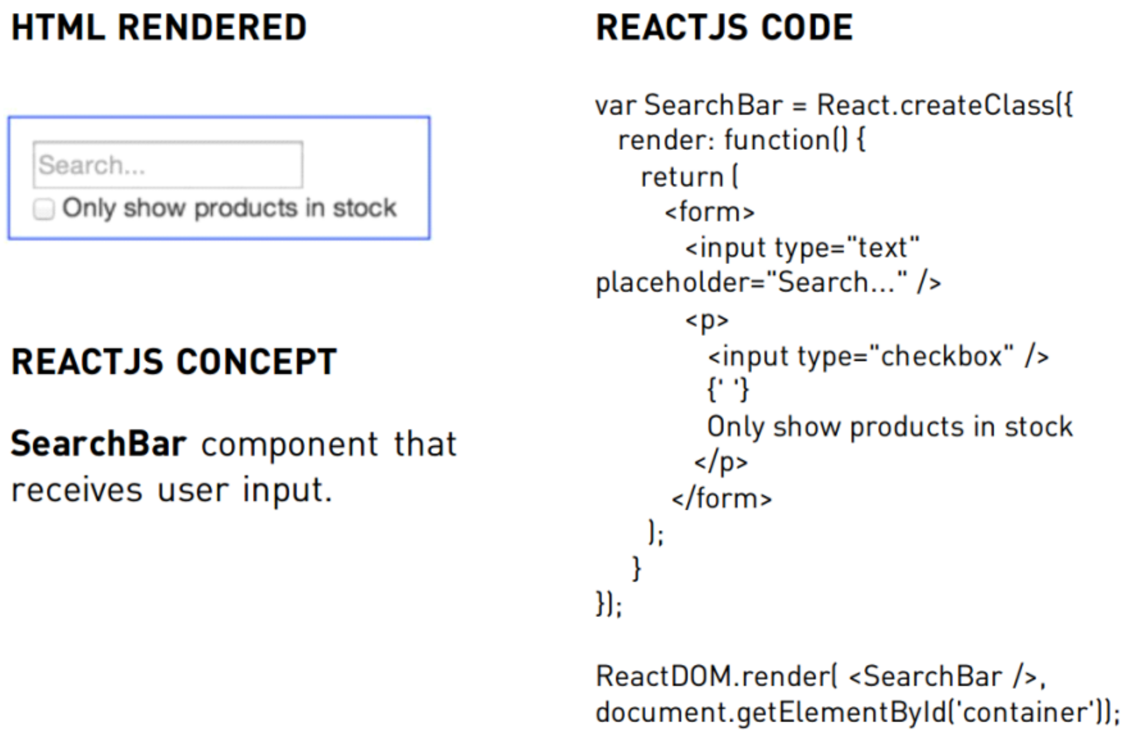
ReactJS vs AngularJS

一些比较，然而我并不知道在说啥 what is MVC? Heavy framework

React.js uses JSX, a strictly-typed object-oriented extension language for JS.

1. Syntax like java with class/ function definition
2. Strict typing for higher productivity and quality code
3. Faster than regular JS and interoperable with JS
4. only supported in Chrome, must compile to JS in ReactJS by render()

As mentioned before, building blocks of ReactJS are called components. It's a JSX(JS) + HTML element rendered through a virtual DOM.



HTML RENDERED

REACTJS CONCEPT

SearchBar component that receives user input.

REACTJS CODE

```
var SearchBar = React.createClass({
  render: function() {
    return (
      <form>
        <input type="text"
placeholder="Search..." />
        <p>
          <input type="checkbox" />
          {' '}
          Only show products in stock
        </p>
      </form>
    );
  }
});

ReactDOM.render( <SearchBar />,
document.getElementById('container'));
```

Two key aspects of ReactJS

Express.js

Database Schema

===

Week 8 Tutorial

Sequelize => MySQL

Handlebars => templating library

===

Week 9

Review:

Display info or mdeia on a website:

Static: html5 css3 canvas
Dynamic: js jquery node.js(php)

Save data in a web readable format:

Flat file: json xml web storage
Database: nosql(mongoDB)/ sql

Access data from a flat file or databse:

Server side: node.js
Client-side: js jquery ajax

Keep track of unique users

Storage: node.js/ html storage/ nosql/ traditional relational databse
Accessing: node.js js jquery

Respond to user action or create interaction

Js jquery

Ensure site works across deivces

Responsive design

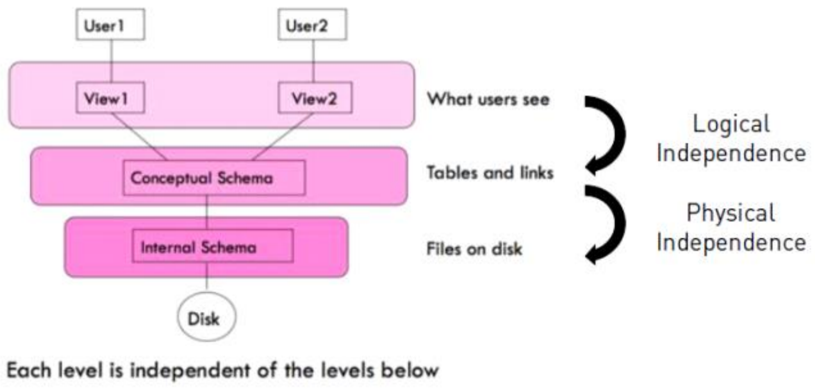
==

Web Architectures

Higher level structures underlying a web app => made up of elements, properties and the relationships among elements. Three levels, each level is independent of the levels below.

Logical independence
Ability to change the logical schema without changing the external schema or app. What users see could be

Physical independence
Ability to change the physical schema without changing the logical shcema



Keep the VIEW(front end) separate from MODEL(back end)

n-tier architectures

Goal: separate the components of a web app into different tiers and layers
Tiers => physical
Layer => logical

1-tier architecture
All 3 layers are on the same machine, and presentation, logic, data are tightly connected

Scalability => single processor means hard to increase volume of processing

Portability => moving to a new machine may mean rewriting everything

Maintenance => changing one layer requires changing the other layers

2-tier architecture
Database runs on a server, but the presentation and logic layer are still tightly coupled
Pros: easily switch to different database
Cons: coupling of presentation and logic leads to heavy server load and network congestion

3-tier architecture
Each tier is independent, can potentially run on different machine.

Presentation layer: front end (no business logic or data accessing code. In web app: it is static or dynamically generated content rendered by the browser (front-end)

Logic layer (middleware, back-end): set of rules for processing the data and can accommodate many individual users, no presentation or data access.
In web app, the logic layer is responsible for dynamic content processing and generation, such as the stuff using Node.js, PHP

Data layer: physical storage for data persistence, manages access to the DB or file system, also called the back-end. No presentation or business logic code.
In web app, the data layer is the database, comprising the data sets and data management system (back-end).

Benefit of 3-tier: easier maintenance, reusable components, faster development (division of work)

Design pattern

Similar to architectures, is a general and reusable approach to a commonly occurring app design problem
尝试解决很多问题，诸如 change the look-and-feel without changing the core logic and/ or data. Present data in different contexts. Interact with/ access data in different contexts/ maintain multiple views of the same data.

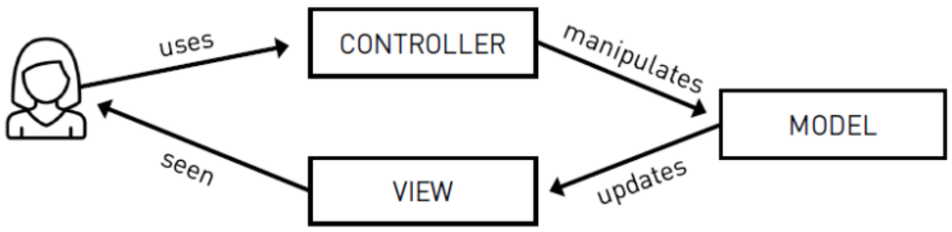
Design pattern: model view controller 对于为何model view controller 三者会各自交互有所疑问，对这个模型不完全理解需要再研究

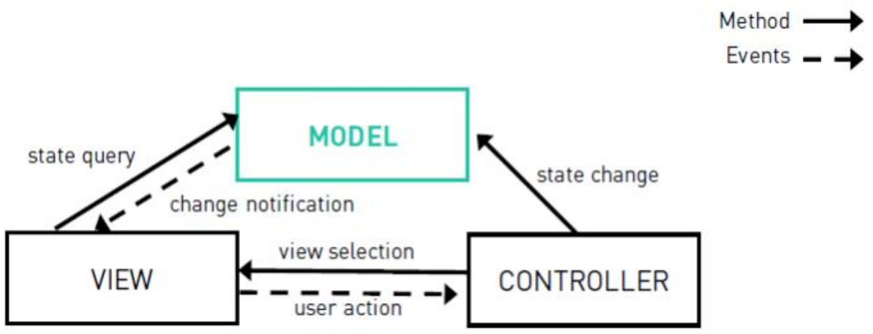
Model: data layer, made up of the database, session information and rules governing transaction. The model manages the application state by responding to requests for information about its state (from the view) or instructions to change state (from controller)
Model is where data lives, 可以是很多形式，, when design a model, we need to think about the structured, relational way about the real world entities. "has-a" => property; "is-a" => belongs to a large group

View: the view renders the model into a form suitable for interaction (user interface). Multiple views can exists for a single model given different contexts. presentation layer, front-end, made up of HTML, CSS and server-side templates(?)
View -> what you see.

Controller: receives user input and initiates a response by making updates to the state of objects in the model and selecting a view for the response. interaction layer, any client-side scripting that manages user inter/actions, HTTP request processing, and business logic/ preprocessing.

> Controller -> smart to figure out what to do (when login to Facebook, there is a controller find you in system, check the right password). Controller has a input and a output



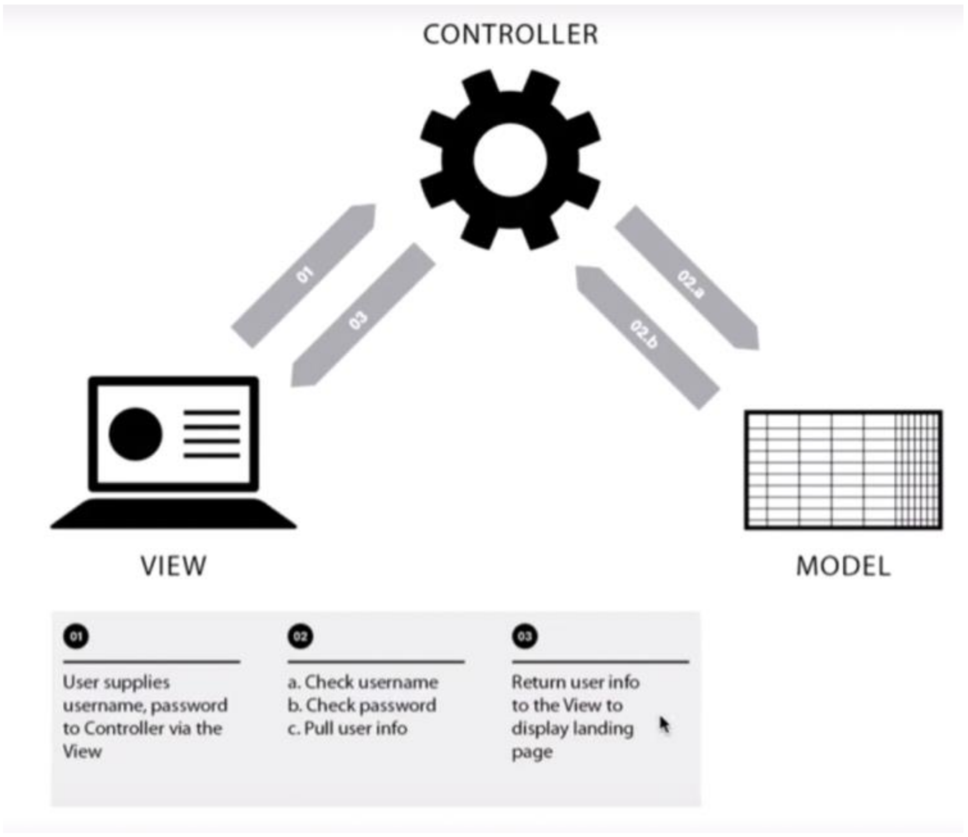


以facebook 举例：

> 当你要登录时: user supplies username, password to controller via the view (user action)
>
> 填入登录信息之后: controller go to model to check username and password, and pull user info
>
> 反馈: controller return user info to the view to display landing page

以angry bird 举例：

> Model includes: varies states of the game like img, layout of the game, rules, state of play
> View includes: the game you experiencing
> Controller includes: decide how bird flies, how the bird collide with the structures it encounters


CONTROLLER
01
03
02.a
02.b
VIEW
MODEL
01
User supplies username, password to Controller via the View
02
a. Check username
b. Check password
c. Pull user info
03
Return user info to the View to display landing page

Benefits of MVC:

Clarity of design/ separation of concerns/ parallel development/ distributable

Two key differences between

Two key difference between 3-tier + MVC

(非常抽象，不能光靠PPT，回头找视频看,要有案例）

Web security

Seven key security questions

Authentication

Identification

Biometrics
False positive: imposter accepted
False negative: authentic user rejected

Two-factor authentication
Two ways of authentication

Authorization

Confidentiality

        - Public key cryptography
    - Data-message integrity

    - Accountability
        - Denial-of-Service attack

    - Availability

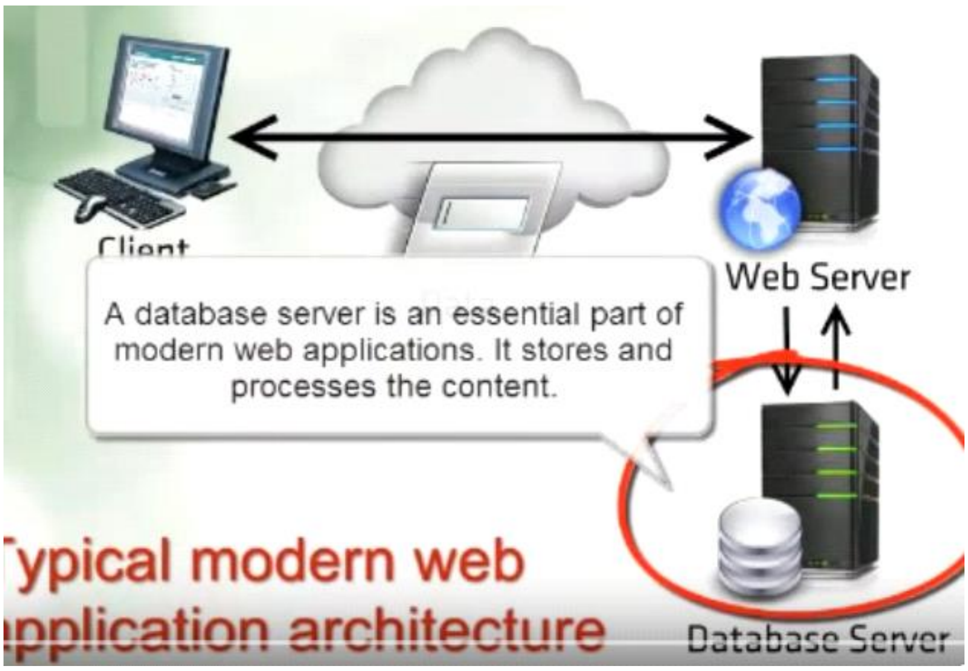
    - Non-repudiation (?)
- Web security threats
        - Phishing 网络仿冒
            - 通过点击链接，进入仿冒站点
        - Pharming 域欺骗
        - Insider threats
        - Click fraud
        - IP spoofing
    - Designing for security
    - Trade-off for users
- Client state manipulation
    - Send a session ID to the client

===

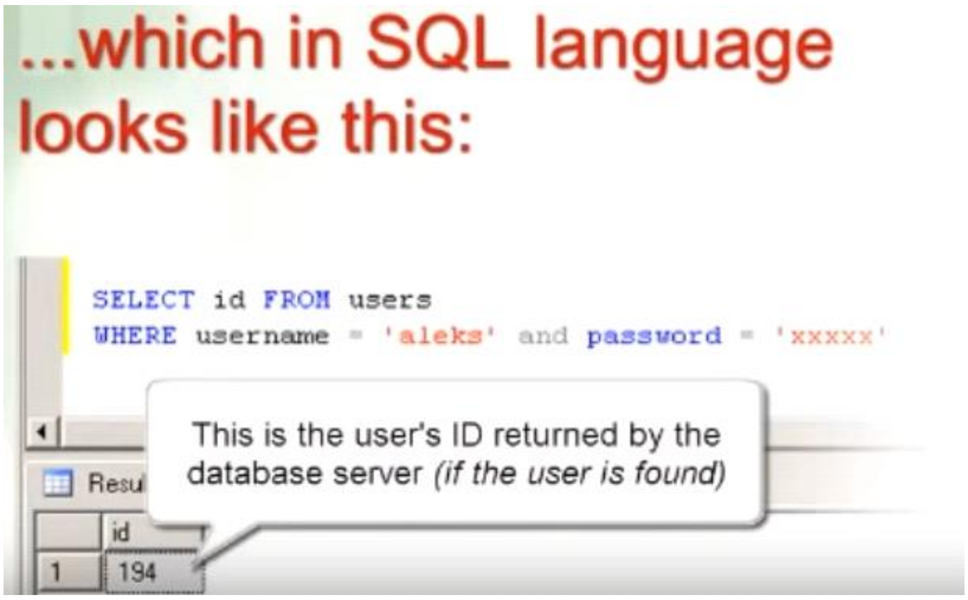
Week 10

SQL injection

    Error based SQL injection


Client
Web Server
A database server is an essential part of modern web applications. It stores and processes the content.
Typical modern web application architecture
Database Server

当你登录网站输入用户名和密码时，前台会问database server是否找到匹配的用户名与密码：



...which in SQL language looks like this:

```
SELECT id FROM users
WHERE username = 'aleks' and password = 'xxxxx'
```

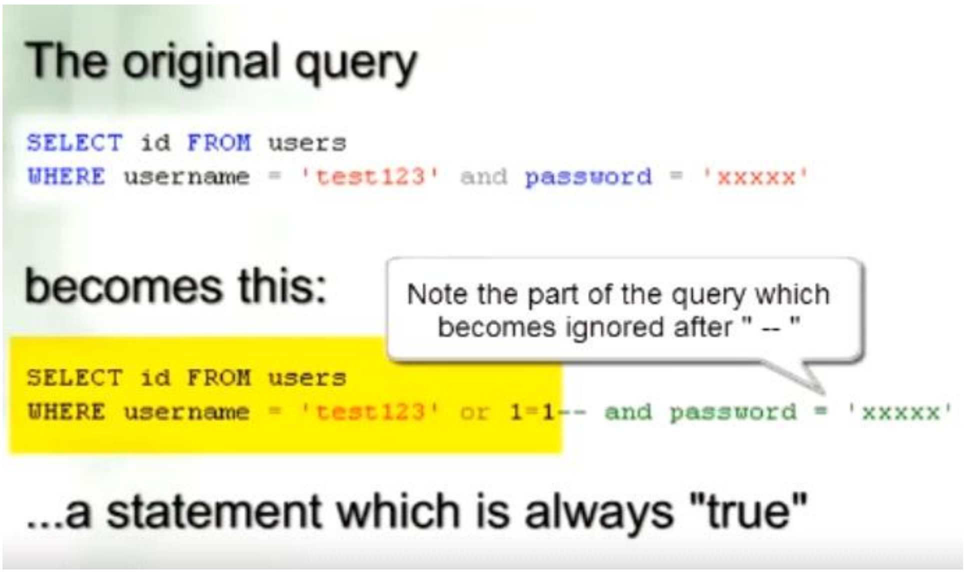
This is the user's ID returned by the database server *(if the user is found)*

	id
1	194

According to SQL syntax, text provided by the user should be always enclosed in single quotes: '____', this single quote char is part of SQL, not user's input. This and some other special characters should always be treated in a special way. 不然这就会出现漏洞, SQL syntax is broken and errror occurs => SQL INJECTION.

Attacker is able to "smuggle" special characters (which is not sanities by web app), it is possible to modify SQL queries, logic, and the application behavior.

如果在用户名中加入特殊符号即可混乱SQL的语法:
Username: test' or 1=1 --



The original query

```
SELECT id FROM users
WHERE username = 'test123' and password = 'xxxxx'
```

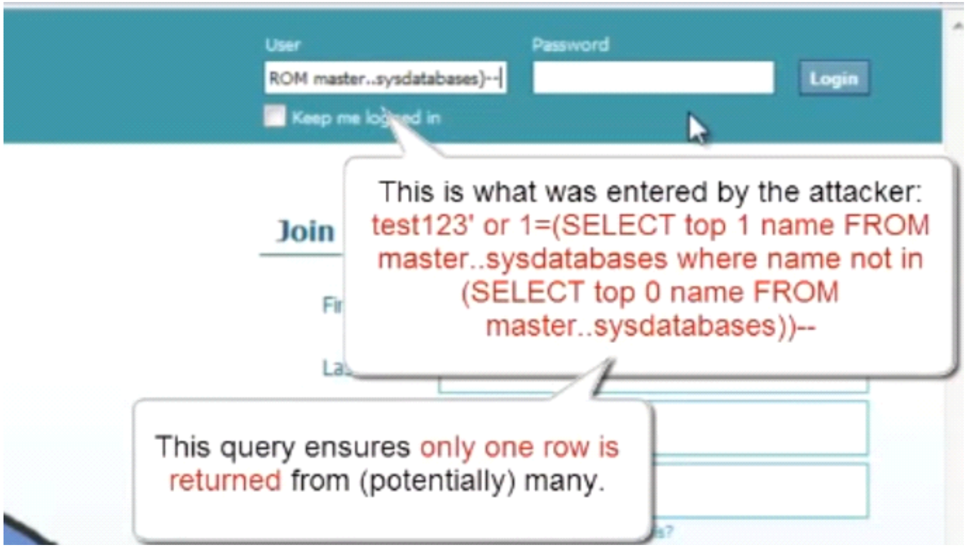
becomes this:

Note the part of the query which becomes ignored after " -- "

```
SELECT id FROM users
WHERE username = 'test123' or 1=1-- and password = 'xxxxx'
```

...a statement which is always "true"

Can do more things like get database version, get all available data



This is what was entered by the attacker: test123' or 1=(SELECT top 1 name FROM master..sysdatabases where name not in (SELECT top 0 name FROM master..sysdatabases))--

This query ensures only one row is returned from (potentially) many.

Extraction of data by using 3rd party tools like Burp Suite

要理解SQL语法

最基础的问题：1 = 1，在SQL的语法中，1=1 always true

MALICIOUS QUERY:

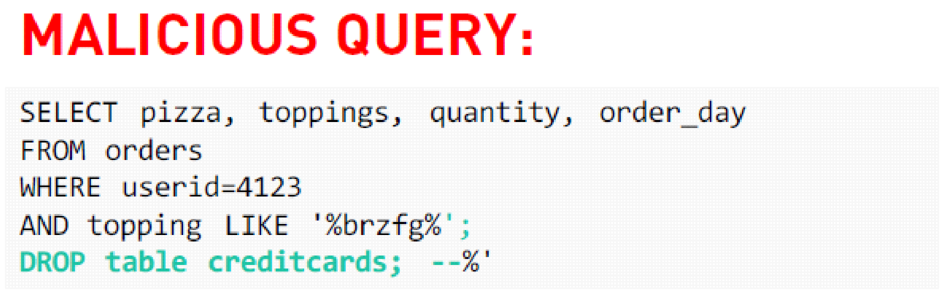
```
SELECT pizza, toppings, quantity, order_day
FROM orders
WHERE userid=4123 AND order_month=0 AND 1=0
UNION SELECT cardholder, number, exp_month, exp_year
FROM creditcards
```

By running this, attack combine two queries, the first table is returned empty/ fails (1 = 0), but the second returns the credit card numbers of all users

MALICIOUS QUERY:

```
SELECT pizza, toppings, quantity, order_day
FROM orders
WHERE userid=4123 AND order_month=0;
DROP TABLE creditcards;
```

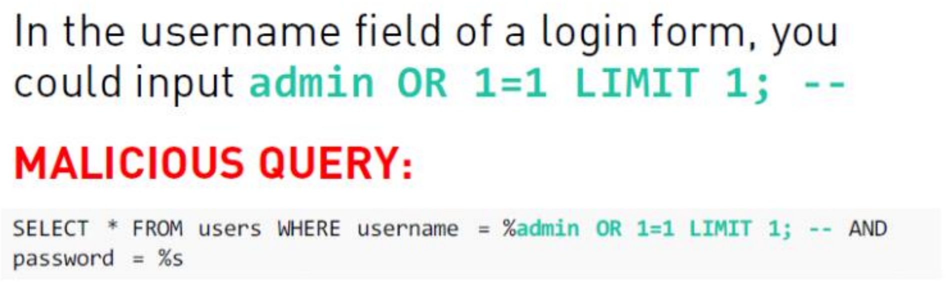
Remove the credit cards table from the database



MALICIOUS QUERY:

```
SELECT pizza, toppings, quantity, order_day
FROM orders
WHERE userid=4123
AND topping LIKE '%brzfg%';
DROP table creditcards; --%'
```

-- is comments in SQL, so it can string comment



In the username field of a login form, you could input `admin OR 1=1 LIMIT 1; --`

MALICIOUS QUERY:

```
SELECT * FROM users WHERE username = %admin OR 1=1 LIMIT 1; -- AND
password = %s
```

By this way, it comments out the password and has an always true statement 1 =1 and get admin access.

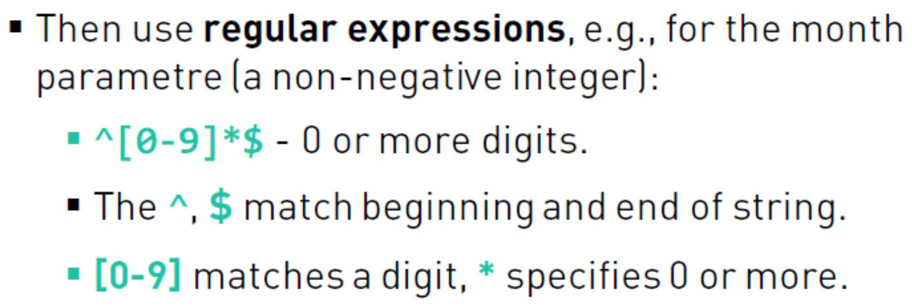
Protect against SQL injection

Whitelisting (instead of blacklisting?)

An instinct strategy => blacklist, filer out some dangerous characters or using built-in SQL functions like kill-quotes() , but:
Cons:

1. May miss some dangerous characters
2. Functions are limited since kill_quotes cannot protect numeric parameters.

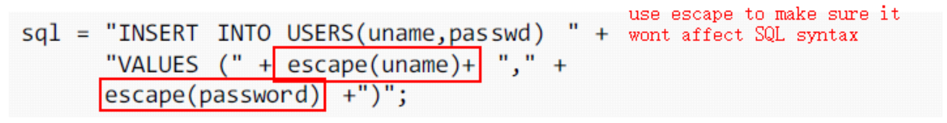
Better way => whitelisting, allow input only if it follows a well-defined set of values by using regular expressions.



- Then use **regular expressions**, e.g., for the month parametre (a non-negative integer):
  - `^[0-9]*$` - 0 or more digits.
  - The `^`, `$` match beginning and end of string.
  - `[0-9]` matches a digit, `*` specifies 0 or more.

Input validation and escaping

Escape quotes with escape() instead of blacklisting.



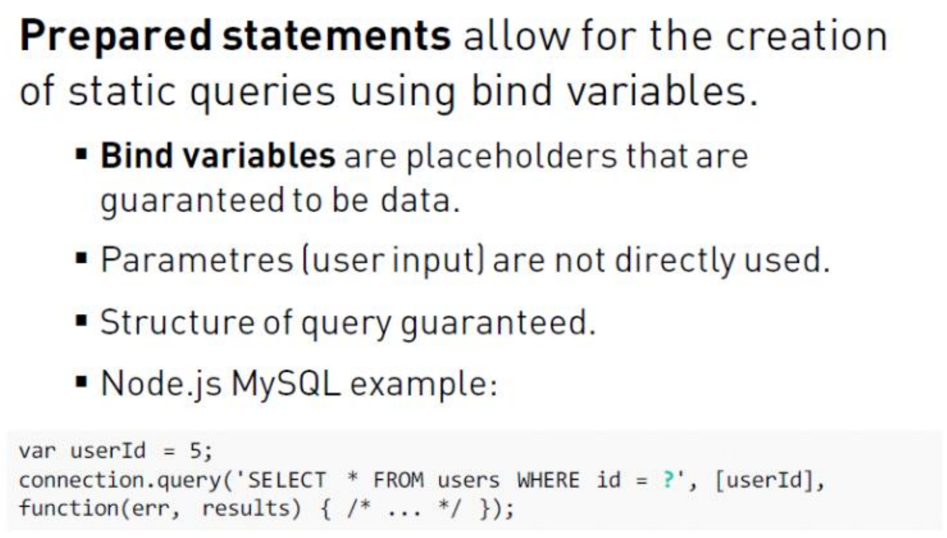
```
sql = "INSERT INTO USERS(uname,passwd) " +
      "VALUES (" + escape(uname)+ "," +
      escape(password) +")";
```

use escape to make sure it wont affect SQL syntax

Cons: this only works for string inputs

Prepared statements and binding variables

Allow for the creation of static queries using bind variables

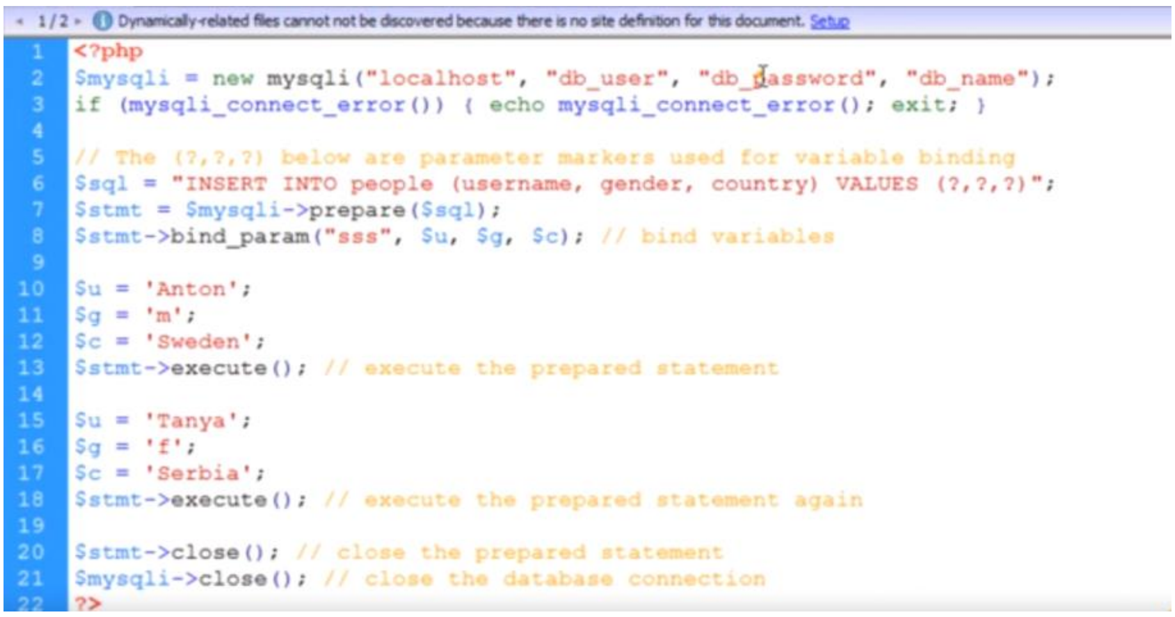


Prepared statements allow for the creation of static queries using bind variables.

- **Bind variables** are placeholders that are guaranteed to be data.
- Parametres (user input) are not directly used.
- Structure of query guaranteed.
- Node.js MySQL example:

```
var userId = 5;
connection.query('SELECT * FROM users WHERE id = ?', [userId],
function(err, results) { /* ... */ });
```

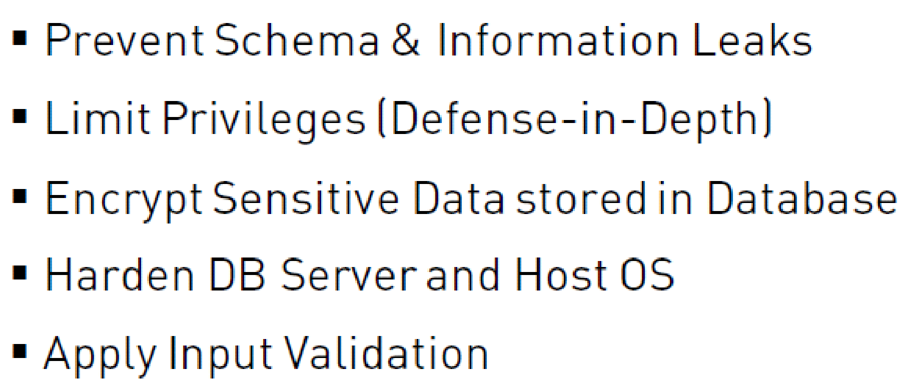
How prepared statements work: 先声明statement的parameters ($sql)， 再对其prepare ($stmt = $mysqli->prepare($sql))，再表明$stmt用来bind。
声明各种变量如$u, $g, $c => 之后执行Bind，最后close



1/2 Dynamically-related files cannot not be discovered because there is no site definition for this document. Setup

```php
<?php
$mysqli = new mysqli("localhost", "db_user", "db_password", "db_name");
if (mysqli_connect_error()) { echo mysqli_connect_error(); exit; }

// The (?,?,?) below are parameter markers used for variable binding
$sql = "INSERT INTO people (username, gender, country) VALUES (?,?,?)";
$stmt = $mysqli->prepare($sql);
$stmt->bind_param("sss", $u, $g, $c); // bind variables

$u = 'Anton';
$g = 'm';
$c = 'Sweden';
$stmt->execute(); // execute the prepared statement

$u = 'Tanya';
$g = 'f';
$c = 'Serbia';
$stmt->execute(); // execute the prepared statement again

$stmt->close(); // close the prepared statement
$mysqli->close(); // close the database connection
?>
```

Measures to mitigate(缓和) the impact



- Prevent Schema & Information Leaks
- Limit Privileges (Defense-in-Depth)
- Encrypt Sensitive Data stored in Database
- Harden DB Server and Host OS
- Apply Input Validation

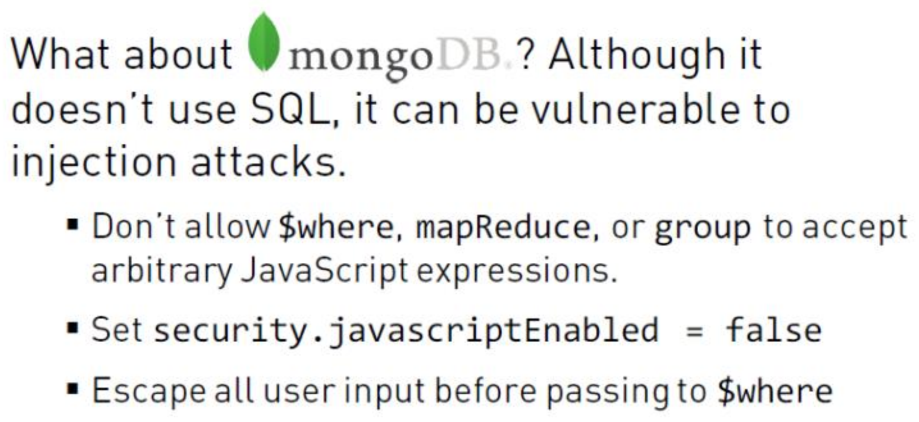
Keep your database schema private because knowing schema make attacker's job easier, don't display error msg and stack traces to external users

Apply the principle of least privilege

Encrypt all sensitive or private data stored in the database one-way or with keys. Don't store the key in the database or attacker could use SQL injection to get it.

Disable unused services and limit accounts since some server have dangerous functions in default

Validation of parameter => not enough, must validate all input at entry point. Reject overly long input. Validation redundancy => use the same validation features on the front end and back end



What about mongoDB.? Although it doesn't use SQL, it can be vulnerable to injection attacks.

- Don't allow `$where`, `mapReduce`, or `group` to accept arbitrary JavaScript expressions.
- Set `security.javascriptEnabled = false`
- Escape all user input before passing to `$where`

```
db.collection.find({
    active: true,
    $where: function() {
        return obj.credits - obj.debits < req.body.input;
    }
});
```

```
db.collection.find({
    active: true,
    $where: function() {
        return obj.credits - obj.debits < 0;
        var date = new Date();
        do {
            curDate = new Date();
        }
        while ( curDate - date < 10000);
    }
});
```

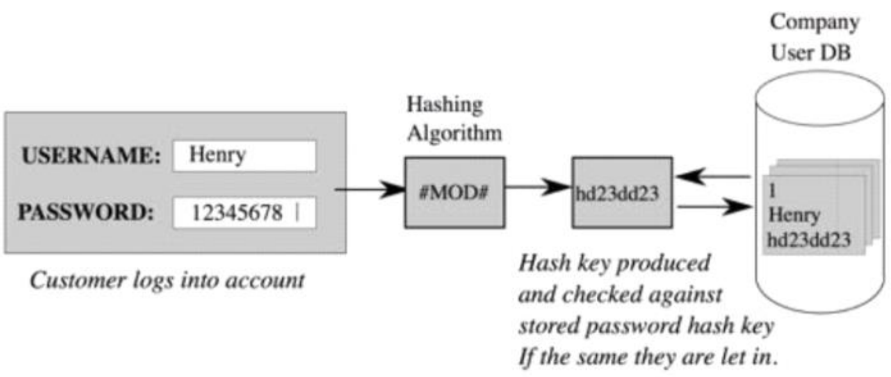
Password Security

- Don't use a flat file/password file.
- **Encrypt** passwords, esp. one-way hashed with MD5 or SHA-1 or similar.

```
john:9Mfsk4EQh+XD2lBcCAvputrIuVbWKqbxPgKla7u67oo=
masako:AEd62KRDHUXW6tp+XazwhTLSUlADWXrinUPbxQEfnsI=
ling:J3mhF7Mv4pnfjcnoHZ1ZrUELjSBJFOo1r6D6fx8tfwU=
```

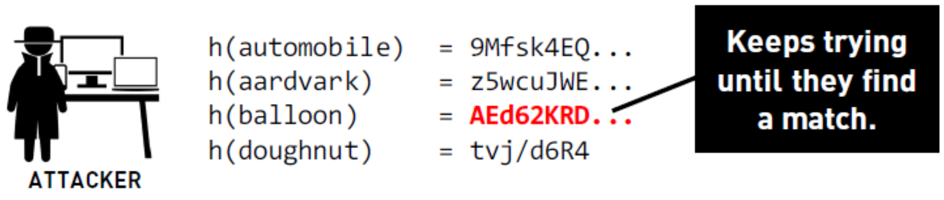
- **Salting** adds an additional random number to the end of each password.

```
john:ScF5GDhWeHr2q5m7mSDuGPVasV2NHz4kuu5n5eyuMbo=:1515
```



Dictionary attack => attacker guess passwords based on word list

- Can be **online** (live system) or **offline** (accesses and download list of user/password combos).



```
h(automobile)   = 9Mfsk4EQ...
h(aardvark)     = z5wcuJWE...
h(balloon)      = AEd62KRD...
h(doughnut)     = tvj/d6R4
```

- Adding salts makes it harder; have to guess it. But if they **target** a user (**chosen victim**), easier.
- And if offline (and patient), greater chance of hack.

如果看到一系列尝试登录但失败的话，可以认定这个IP有问题并封IP

强制要求用户使用强度很高的密码

网站设立“诱饵”，及时提醒管理员

要求用户经常更换密码

建议用户的密码为可以被读出来的，但不在字典中的字（因为有dictionary attack）

如果用户几次登录失败，人工设立再次登录等待时间

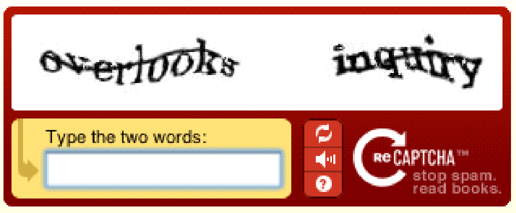
给用户提供每次登录的信息（如时间地点），让用户自己发现是否被盗

To combat phishing（网站仿冒），let user select an img rather than click on link

CAPTCHA system:

CAPTCHA: Telling Humans and Computers Apart Automatically

A CAPTCHA is a program that protects websites against bots by generating and grading tests that humans can pass but current computer programs cannot. For example, humans can read distorted text as the one shown below, but current computer programs can't:



One-time passwords: works but annoying for returning frequent users

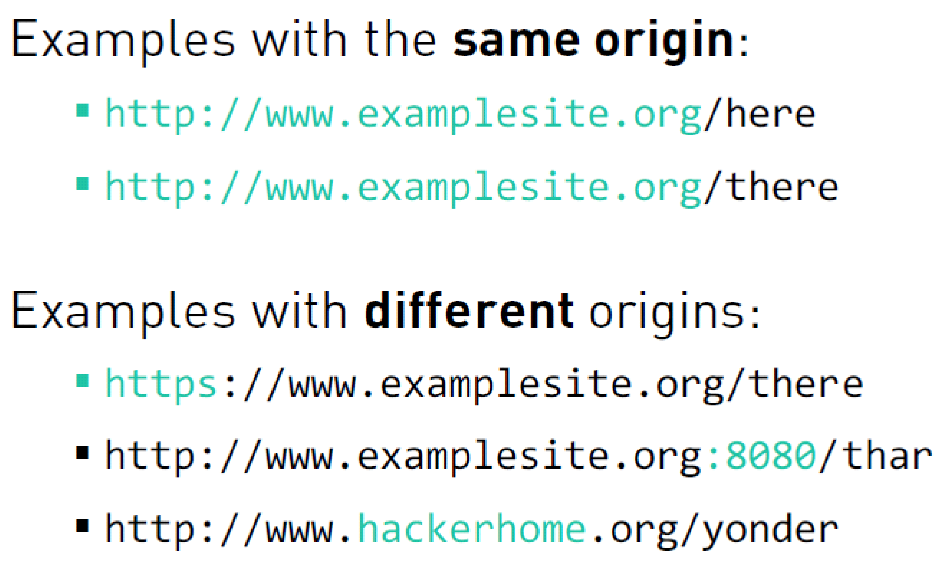
Passphrases: sentence instead of word

Cross-domain security

Same-origin policy => scripts can only access data from other sites if they have same origin.

- Where **origin** is a combination of the protocol, hostname, and port number.
- Hyperlinks, embedded (i)frames, data inclusion (GET/POST, JSONP) across domains still possible.

```
<iframe style="display: none"
src="http://www.mywwwservice.com/some_url"></iframe>
```



Examples with the **same origin**:

- http://www.examplesite.org/here
- http://www.examplesite.org/there

Examples with **different** origins:

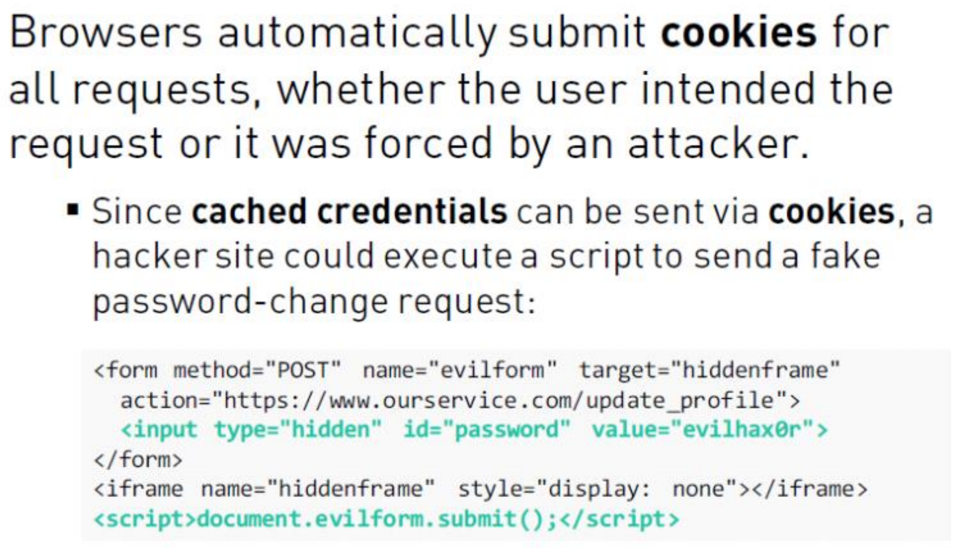
- https://www.examplesite.org/there
- http://www.examplesite.org:8080/thar
- http://www.hackerhome.org/yonder

Hijack cookies

Security issues from browser interacting with multiple web apps

Cross-site request forgery (XSRF)

Malicious sites initiates HTTP requests to app on a user's behalf but without knowledge. It works because same-origin policy allows sending of GET/POST requests to different origins



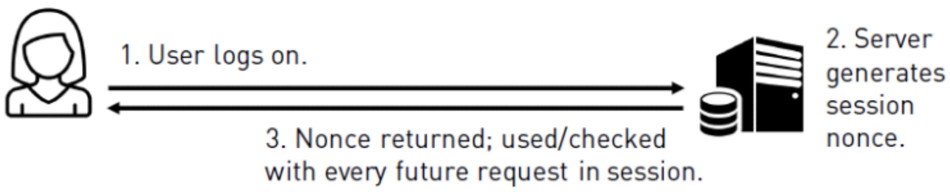
Browsers automatically submit **cookies** for all requests, whether the user intended the request or it was forced by an attacker.

- Since **cached credentials** can be sent via **cookies**, a hacker site could execute a script to send a fake password-change request:

```
<form method="POST" name="evilform" target="hiddenframe"
  action="https://www.ourservice.com/update_profile">
  <input type="hidden" id="password" value="evilhax0r">
</form>
<iframe name="hiddenframe" style="display: none"></iframe>
<script>document.evilform.submit();</script>
```

The best way to protect against XSRF attacks is to use **secret tokens** (or **nonces**).

- Design your app so that the token submits along with the cookie on following requests.
- Since attackers can't guess the token, you can **check** if the user wanted to send a request or not.


1. User logs on.
2. Server generates session nonce.
3. Nonce returned; used/checked with every future request in session.

Cross-site script inclusion (XSSI)

In a **cross-site script inclusion (XSSI)** attack, a third party can include a `<script>` sourced from our web app. Two flavours:

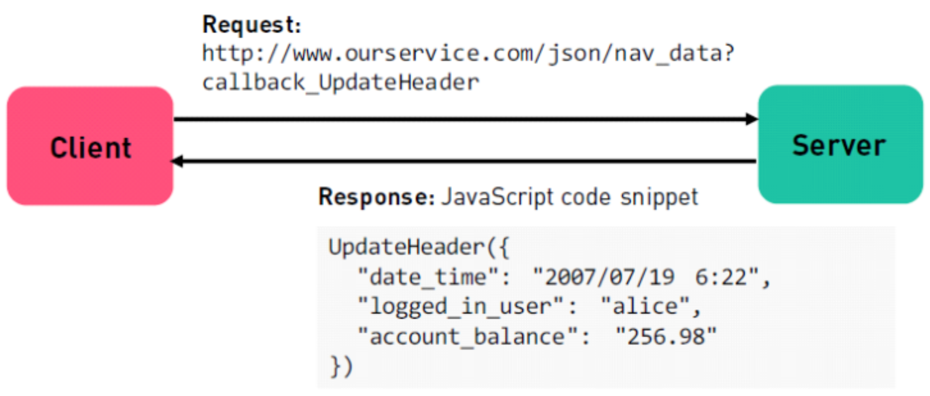
Static script inclusion, where the purpose is to enable code sharing, e.g., jQuery library hosted on Google. This **dangerous** because it runs in our app with full access to client data.

Dynamic script inclusion, where instead of a traditional postback (HTTP POST) of a new HTML page, asynchronous requests (AJAX) are used to fetch data.

Say that you have the following:

```
<script src="http://yourapp.com/secret"></script>
```

- If http://yourapp.com/secret executes JS code that returns **sensitive data** or **functions**, this data can be **stolen** and the functions can be **replaced** with the attacker's version!
- If http://yourapp.com/secret returns a **JSON array,** an attacker could **override** the array **constructor** to steal the array's contents.


Request:
http://www.ourservice.com/json/nav_data?callback_UpdateHeader
Client
Server
Response: JavaScript code snippet

```
UpdateHeader({
  "date_time": "2007/07/19 6:22",
  "logged_in_user": "alice",
  "account_balance": "256.98"
})
```

Protect from XSSI attacks

- Check the origin and referer **headers** (if possible).
- Use **XSRF tokens**, which are randomly and securely generated on the server-side and included as a hidden field. On the server-side, they are required for **change operations** and should be **validated**.
- Don't include sensitive data in shared scripts.
- Use **sanitizer** with Node.js apps (Google's Caja).

Cross-site scripting (XSS) => attacker bypasses the same-origin policy by executing a malicious script in app

==

Week 11

Unit testing (JS)

Test small pieces of JS code. In nodejs, we test both front end and back end. Tools: nodeunit and mocha.

Nodeunit only test back-end node.js js code.

In /lib/arrayfun.js:

```
exports.indexOf = function(arr, i) {
  return arr.indexOf(i);
}
```

In /test/test.js

```
var arrayfun = require('../lib/arrayfun');

exports['indexOf'] = function (test) {
  test.equal(arrayfun.indexOf([1,2,3], 5), -1);
  test.done();
};
```

```
  test.equal(arrayFun.indexOf([1,2,3], 5), -1);
  test.done();
};
```

Mocha can test both front and back end'

```
var assert = require('chai').assert;

describe('Array', function() {
  describe('#indexOf()', function() {
    it('should return -1 when the value is
not present', function() {
      assert.equal(-1, [1,2,3].indexOf(5));
      assert.equal(-1, [1,2,3].indexOf(0));
    });
  });
});
```

See the code example

Performance on the front end

Different ways to tackle performance:

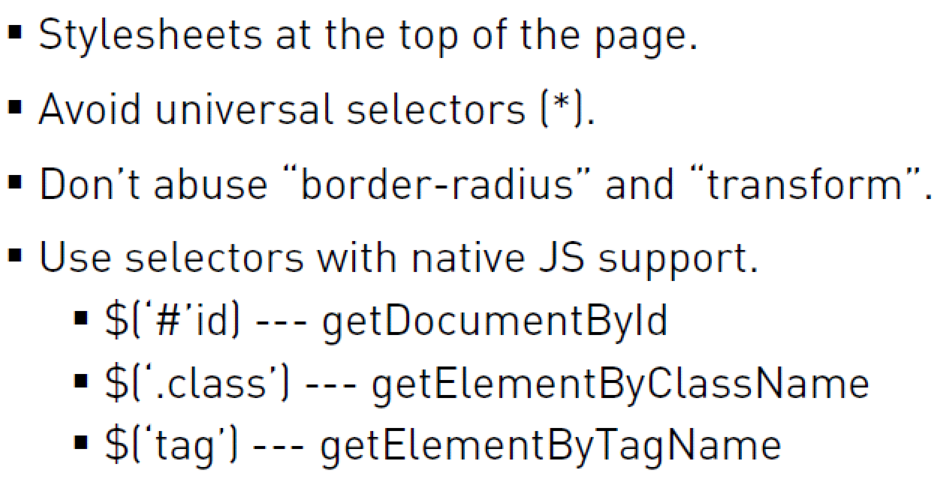
Design strategies:

- Use CSS (e.g., gradients) instead of bg images.
- Lazy image loading: Placeholder; loading triggered when you scroll to that point in the page.
- Customize plugins and libraries so that you only use what capabilities you need, e.g. you can download only what you need with libraries like Modernizr and jQuery .
- Render core elements first.

Reflow: browser process of recalculating the positions and geometries of elements in the DOM because of resizing window or using JS methods that manipulate the DOM.(每次调整页面大小或者加新东西时刷新页面的感觉)

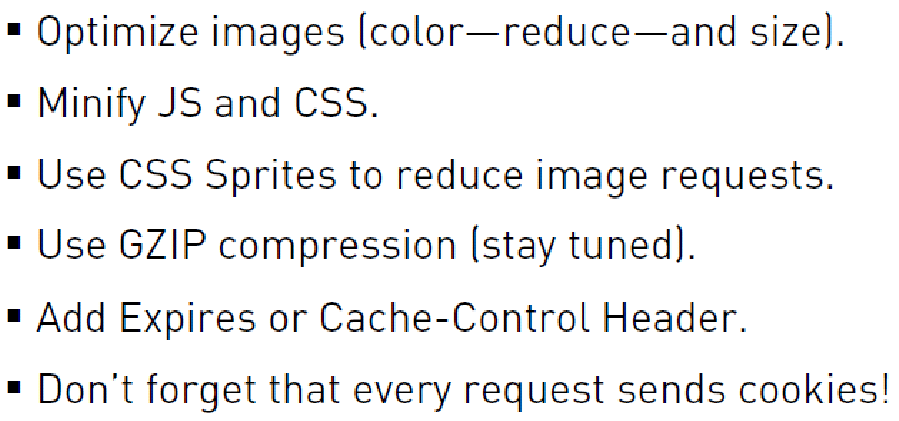
Minimize reflow:
Reduce unnecessary DOM depth.
Reduce number of CSS rules used
Do complex procedures outside of the rule; make use of position: absolute/ fixed
Avoid overly complicated CSS selectors
Lazy load content and use idle time to load content

CSS techniques for performance:

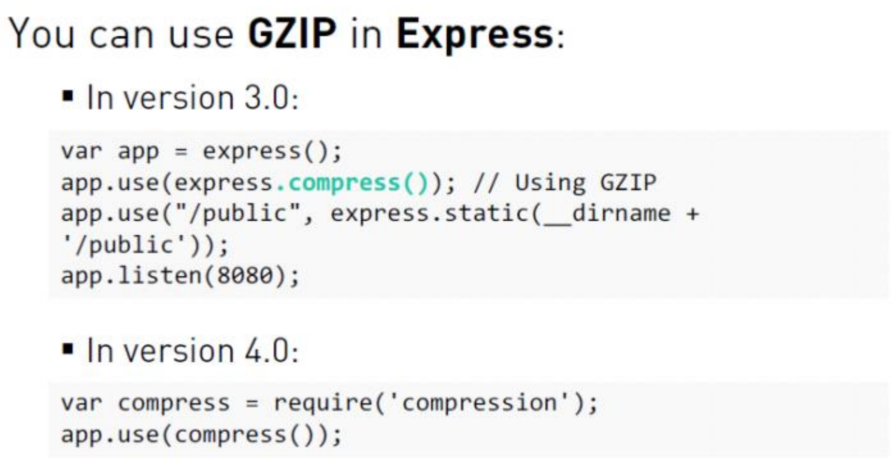


- Stylesheets at the top of the page.
- Avoid universal selectors (*).
- Don't abuse "border-radius" and "transform".
- Use selectors with native JS support.
  - $('#'id) --- getDocumentById
  - $('.class') --- getElementByClassName
  - $('tag') --- getElementByTagName

Front-end optimization:



- Optimize images (color—reduce—and size).
- Minify JS and CSS.
- Use CSS Sprites to reduce image requests.
- Use GZIP compression (stay tuned).
- Add Expires or Cache-Control Header.
- Don't forget that every request sends cookies!



You can use **GZIP** in **Express**:

- In version 3.0:

```
var app = express();
app.use(express.compress()); // Using GZIP
app.use("/public", express.static(__dirname +
'/public'));
app.listen(8080);
```

- In version 4.0:

```
var compress = require('compression');
app.use(compress());
```

Network-wise:

Since each page invokes many HTTP requests and resources. In order to improve performance:

- Make fewer HTTP requests.
- Content Delivery Network (e.g., akamai).
- Split resources across servers (load balance).
  - But avoid too many DNS lookups.
- Limit cookies, which can consume bandwidth.
- Be careful with redirects (301, 302).

HTTP/2 => HTTP persistent connection

Use single TCP connection to send and receive multiple http requests/response.

- Lower CPU and memory usage (because fewer connections are open simultaneously).
- Enables pipelining of requests and responses.
- Reduced network congestion (fewer TCP con.).
- Reduced latency in subsequent requests (goodby handshaking).

HTTP headers

HTTP headers allow the client and the server to pass additional information with the request or the response. A request header consists of its case-insensitive name followed by a colon ':', then by its value (without line breaks). Leading white space before the value is ignored.

- You can tell the browser to cache **specific resources** from your application, e.g., images, JS files, CSS files, etc.
- The **benefits** are:
  - Reduction in latency: Client will receive response faster.
  - Reduction in network bandwidth: Network can handle more messages.

Here's some **useful** HTTP headers:

- `max-age=[seconds]`: Set max amount of time a resource will be considered fresh.
- `no-cache`: Forces server validation before releasing a cached copy.
- `no-store`: Disables caching a resource.
- `public/private`: Manages caching a response to a public (shared) cache.
- `Last-Modified` or `ETags`: Cache tries to validate its older copy against your app's.

In **Express** when using **static** files, you can make use of the **maxAge** parameter:

```
app.use(express.static(__dirname + '/public', {
        maxAge: 86400000
}));
```

And a **favicons** example:

```
var favicon = require('serve-favicon');
app.use(favicon(path.join(__dirname, 'public',
'favicon.ico'), {
        maxAge: 86400000
}));
```

Performance on the back-end

Node is asynchronous and single-threaded, yet almost all computers are **muti-core** now.

We can make use of this setup for **load balancing**: one Node process per core.

We'll use the **cluster** & **http-proxy** modules.